Numerical Computation of Newton's Method

We shall discuss how to solve Nonlinear Systems of Equations using Newton's method. We will further demonstrate how to compute the Jacobian using automatic differentiation. Convergence results of Newton's method if any exist for our problem shall be discussed. All these shall be done in Julia.jl.

- NB: Reference is herein made to the Scientific Computing Lecture by Dr. Jurgen Fuhrmann (WS $21/22$) for some coding idea.

Basic Idea

This method is basically based on the idea of linear approximation. The tangent line at a given point is a good approximation of the function. Newton's iteration approximates a function based on its tangent line, then takes the tangent line's zero as the next guess ϕ_i.

We begin with an initial (suitable) guess ϕ_0. Then, ϕ_0 is used in to produce ϕ_1 and ϕ_1 is further used to produced ϕ_2 and so on. The objective is that one will ultimately find a value ϕ_i that is close enough to the desired solution.

Recall the Taylor Series approximation of polynomials of increasing powers;

$$f(\phi) = f(\phi_0) + (\phi - \phi_0)f'(\phi_0) + \frac{1}{2}(\phi - \phi_0)^2 f''(\phi_0) + \dots$$

where f' and f'' are the first and second derivative of f at ϕ_0. If we consider the first two terms of the Taylor series expansion we have:

$$f(\phi) = f(\phi) + f'(\phi_0)(\phi - \phi_0)$$

setting $f(\phi) = 0$,

$$0 = f(\phi_0) + (\phi - \phi_0)f'(\phi_0)$$

$$(\phi - \phi_0)f'(\phi_0) = -f(\phi_0)$$

$$(\phi - \phi_0) = \frac{-f(\phi_0)}{f'(\phi_0)}$$

$$(\phi - \phi_0) = \frac{-f(\phi_0)}{f'(\phi_0)}$$

In general, Newton's iteration in 1D is given by

$$\phi_{i+1} = \phi_i - \frac{f(\phi_i)}{f'(\phi_i)}$$

- **In multidimension:**

We extend Newton's iteration in 1D into multi-dimensional system.

Essentially, we are finding the zeroes of $F(\phi) = b$

where, $F(\phi) := A\phi + \Psi(\phi)$ and $b = 0$.

In this case, $F(\phi)$ is regarded as a nonlinear operator $F : D \longrightarrow \mathbb{R}^n$ where $D \subset \mathbb{R}^n$ is the domain of definition, $\phi = [\phi_1, \phi_2, \dots, \phi_n]^T \in \mathbb{R}^n$ and $b = [0, 0, \dots, 0]^T \in \mathbb{R}^n$ is a zero vector.

The scalar ϕ_i in the 1D case is replaced by vector ϕ_i and we left multiply $F(\phi_i)$ by the inverse of it's Jacobian matrix $F'(\phi_i) = J(\phi_i)$.

This results in a multidimensional Newton's iteration equation:

$$\phi_{i+1} = \phi_i - J(\phi_i)^{-1}F(\phi_i)$$

The inverse of $F(\phi)$ is expensive to compute. Hence instead of computing $J(\phi)^{-1}$, we solve a linear equation by rearranging the iteration eqaution

$$J(\phi_i)\delta_i = -F(\phi_i)$$

where $\delta_i = \phi_{i+1} - \phi_i$ using an initial guess $\phi_0 \in \mathbb{R}^n$ and $\delta_i \in \mathbb{R}^n$.

we can then find δ_i such that,

$$\delta_i = -J(\phi_i)^{-1}F(\phi_i)$$

.

Hence we replace $-J(\phi_i)^{-1}F(\phi_i)$ with δ_i in the iteration equation such that

$$\phi_{i+1} = \phi_i - \delta_i$$

Step one : Inital Vector

The first step of Newton's method is to determine an initial vector. A standard way of determining an initial vector will be to compute the intersection of all three functions. Let ϕ_0 be our initial vector,

$$\phi_0 = [0.1 \quad 0.1 \quad -0.1]^T$$

and let ϕ^* be the solution to the system $F(\phi) = b$.

```
ϕ₀ = [0.1, 0.1, -0.1]
```

```
ϕ₀ = [0.1,0.1,-0.1]
```

Step Two : Define the System

Next, we define the components of the nonlinear system $F(\phi) = b$. We define first the left hand vector $F(\phi)$ followed by the right hand vector b, which is a zero vector.

```
F (generic function with 1 method)
```

```
F(ϕ) = G(ϕ)
```

```
G (generic function with 1 method)
```

```
function G(ϕ)
    [3*ϕ[1] - cos(ϕ[2]*ϕ[3])- 1/2;
    ϕ[1]^2 - 81*(ϕ[2] + 0.1)^2 + sin(ϕ[3]) + 1.06;
    exp(-ϕ[1]* ϕ[2]) + 20*ϕ[3] + (10*pi - 3)/3]
end
```

```
b = [0, 0, 0]
```

```
b = [0,0,0]
```

Step Three : Evalaute F at ϕ_0

Now, we evaluate $F(\phi)$ at the inital vector ϕ.

```
[-1.19995, -2.26983, 8.46203]
```

```
F(ϕ₀)
```

After evaluating $F(\phi_0)$, we obtained a row vector. By transposing the row vector, we have the following column vector:

$$F(\phi_0) = \begin{bmatrix} -1.19995 \\ -2.26983 \\ 8.46203 \end{bmatrix}$$

Step four : Residual

Here, the goal is to obtain the residual of the system. We compute the residual at the inital vector ϕ_0.

```
▸ [-1.19995, -2.26983, 8.46203]
```

```
begin
    r(ϕᵢ) = F(ϕᵢ) - b    #i = 0,...n
    r(ϕ₀)
end
```

We transpose the compute the residual above and obtain the following column vector

$$r(\phi_0) = \begin{bmatrix} -1.19995 \\ -2.26983 \\ 8.46203 \end{bmatrix}$$

Step five : Jacobian

We compute and evaluate the Jacobian of $F(\phi)$ at ϕ_0. We denote the Jacobian at ϕ_0 by $J(\phi_0)$.This is done using Julia's automatic differetiation package "ForwardDiff: jacobian".

To begin with, we create a result buffer for our 3×3 nonlinear system in order to use automatic differentition to compute the Jacobian. We use the DiffResults.JacobianResult package. This is saved as d_results.

Next, we call the ForwardDiff.jacobian! package on F with the inital vector ϕ_0 as input data. This mutates the $d_{results}$. Additionally, within the $d_{results}$, we otain the value for the differential results. Finally, we obtain the Jacobian at ϕ_0, which is a 3×3 matrix as well as it's inverse $J^{-1}(\phi_0)$.

```
▸ [-1.19995, -2.07017, 12.462]
```

```
begin
    d_result=DiffResults.JacobianResult(ϕ₀)
    ForwardDiff.jacobian!(d_result,F,[0.1, 0.1, 0.1])
    DiffResults.value(d_result)
end
```

```
3×3 Matrix{Float64}:
  3.0         0.000999983   0.000999983
  0.2       -32.4           0.995004
 -0.099005   -0.099005     20.0
```

```
begin
    J(ϕ₀) = DiffResults.jacobian(d_result)
    J(ϕ₀)
end
```

```
3×3 Matrix{Float64}:
 0.333332     1.03404e-5   -1.71808e-5
 0.0021086   -0.0308688     0.00153563
 0.00166051  -0.000152757   0.0500075
```

```
inv(J(ϕ₀))
```

The results are written explicitly below, where the Jacobian is given by:

$$J(\phi_0) = \begin{bmatrix} 3.0 & 0.000999983 & 0.000999983 \\ 0.2 & -32.4 & 0.995004 \\ -0.099005 & -0.099005 & 20.0 \end{bmatrix}.$$

and the inverse of the Jacobian is given as,

$$J(\phi_0)^{-1} = \begin{bmatrix} 0.333332 & 1.03404e-5 & -1.71808e-5 \\ 0.0021086 & -0.0308688 & 0.00153563 \\ 0.00166051 & -0.000152757 & 0.0500075 \end{bmatrix}.$$

Solving the System

All the above steps are brought together in the Newton function below. The function below is called on F, b and the initial vector ϕ_0 at a tolerance level of $1.0e-12$ for an iteration limit of 100.

```
Newton (generic function with 1 method)
```

```
function Newton(F,b,ϕ₀; tol=1.0e-12, maxit=100)
    result=DiffResults.JacobianResult(ϕ₀)                    #result buffer for Jacobian
    history=Float64[]                                        #history vector
    ϕ=copy(ϕ₀)
    it=1
    while it<maxit
        ForwardDiff.jacobian!(result,(ϕ)->F(ϕ) - b,ϕ)
        res=DiffResults.value(result)                         #resuidual reuslt
        jac=DiffResults.jacobian(result)                      #Jacobian result
        δ=jac\res                                             #solve for δ
        ϕ= ϕ - δ                                             #update
        nm=norm(δ)                                           #norm
        push!(history,nm)                                    #push norm to history
        if nm<tol
            return ϕ,history
        end

        it=it+1
    end
    throw("convergence failed")
end
```

For our outcomes, we have the following functions:

- the approximate solution obtained after the iteration is given by the function Newton_result
- Newton_history shows results for convergence
- And residual for the residual results.

```
(
  1: [0.5, 8.10963e-18, -0.523599]
  2: [0.586567, 0.0179945, 0.00157676, 1.24488e-5, 7.76083e-10, 8.90637e-17]
  3: [0.0, -2.22045e-16, 1.77636e-15]
)
```

```
begin
    Newton_result,Newton_history=Newton(F,b,ϕ₀)
    residual = F(Newton_result)-b
    Newton_result,Newton_history,residual
end
```

Convergence Result

The measure of convergence is checked by computing the norm $||\phi_{i+1} - \phi_i||$ in every iteration step and the tolerance level.

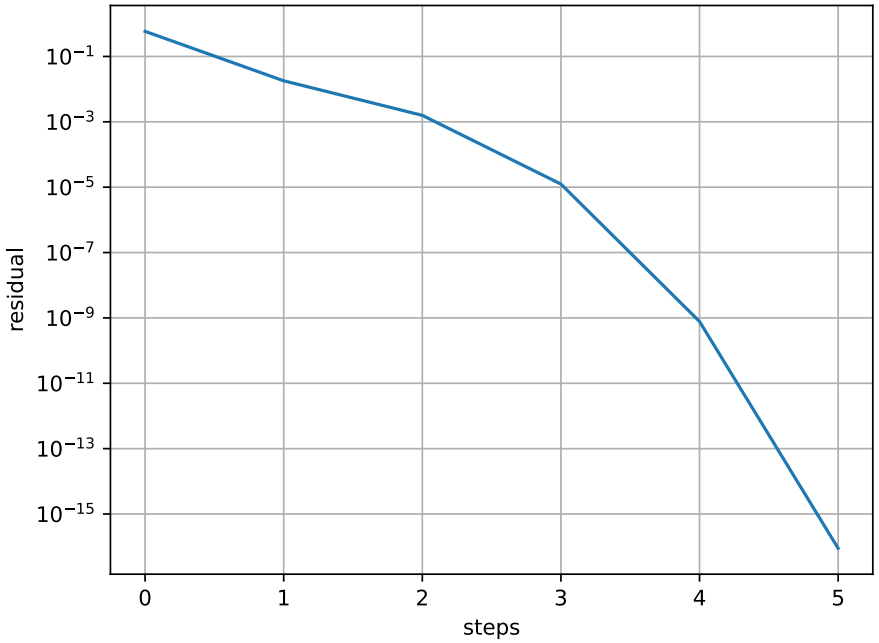
Once the first approximate solution vector ϕ_1 is computed, the iteration process continous until we obtain convergence to the solution ϕ^*.

For convergence of a set of vectors, the norm must be zero. That is,

$$||\phi_{i+1} - \phi_i|| = \sqrt{(\phi_{i+1}^1 - \phi_i^1)^2 + \ldots + (\phi_{i+1}^n - \phi_i^n)^2} = 0.$$

Therefore, Newton's iteration function will terminate when the norm is zero and less than the level of tolerance.

The convergence of our iteration is visualized below. We observe from the below that, in fewer iteration steps, we have a good behaviour of the residual. That is, in $\mathbf{6}$ steps, we obtain a quadratic convergence which is desirable.



Discussion

Iteration steps: 6

Using the Newton function, we obtained the result for our system $F(\phi) = b$ in six iteration steps. The iteration stopped due to the fact that, we obtained convergence as explanied earlier.

Thus, the approximate solution of the system from our computation is written explicitly as:

$$\phi^* = \begin{bmatrix} 0.5 \\ 8.10963e-18 \\ -0.523599 \end{bmatrix}$$